FINAL EXAM with SOLUTION

1. Points MAY be taken off for failure to comply with general instructions.
2. Use ONLY the front side of each page for your answers.
3. Use the REVERSE SIDE of each page for scratch work.
4. Please answer ALL questions.

GENERAL COMMENTS ABOUT YOUR ANSWERS:

- *The usual...*

1. SHORT QUESTIONS
Your answers for this question MUST only be written on the provided space in the question sheet. Please use the back of this question sheet for scratch work; only use exam booklets if you need extra space.

(0) PLEASE WRITE YOUR NAME HERE.

NAME:____

(i) AVL TREES (10 Points)
Draw two AVL trees, with n keys each: the two trees must have different heights. Make n as small as you can.

ANSWER:

A:

A:

A:

SOLUTION:
The smallest possible value of n is $n = 7$. See the trees in Figure 1.

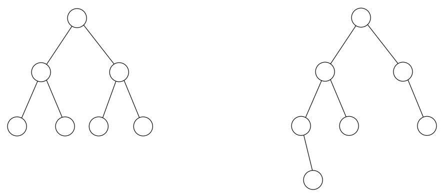

Figure 1: AVL Trees with 7 nodes, with heights 2 and 3, respectively

(ii) HASHING (15 Points)
In the multiplication method for hash functions, we can use any irrational number $\alpha > 0$. Suppose the values of $\{\alpha\}, \{2\alpha\}, \{3\alpha\}$ to three decimal places, and given in sorted order, are $0.243 < 0.414 < 0.828$. Recall that $\{\alpha\}$ is the fractional part of a number α. What are the possible values for $\{4\alpha\}$?

ANSWER:

A:

A:

A:

SOLUTION:
The first three insertion values produces four subintervals, say $I_0 < I_1 < I_2 < I_3$. Their lengths (to 3 decimal places) are $0.243, 0.169, 0.414, 0.169$. According to the 3 Distances

Theorem so V. Sós, the next insertion $x = \{4\alpha\}$ must split the longest subinterval, namely I_2 with length 0.414. Since $I_2 = [0.414, 0.828]$, the new lengths after this insertion are $x - 0.414$ and $0.828 - x$. But at least one of these two lengths must be equal to the 0.243 and 0.169 (otherwise we have 4 distinct distances). It follows that $x = 0.414 + 0.243 = 0.657$ or $0.414 + 0.169 = 0.583$. REMARK: actually $\alpha = \sqrt{2}$ in this example, and $x = \{4\alpha\}$ turns out to be 0.657.

(iii) LCS (15 Points)
In the Longest Common Subsequence (LCS) problem, show the inequality $L(XX, Y) \leq 2L(X, Y)$. where X, Y are any two strings.

`ANSWER:`____

`A:`____

`A:`____

`A:`____

SOLUTION:
Let $Z \in LCS(XX, Y)$. Let $\ell = |Z| = L(XX, Y)$.

Here are two proofs from students:

Michael Haag: Let $L(XX, Y) = m + n$, where m of the first X, and n of the second X, are used in an optimal solution in $LCS(XX, Y)$. Then $L(X, Y) \geq \max\{m, n\}$. Hence $2L(X, Y) \geq m + n = L(XX, Y)$.

Yanjun Wang: Let $Z \in LCS(X, Y)$. Then $L(XX, Y) \leq L(XX, Z) + L(Z, Y)$ by triangular inequality. But $|Z| = L(XX, Z) = L(Z, Y) = L(X, Y)$. Hence $L(XX, Y) \leq 2L(X, Y)$.

(iv) PROBABILISTIC ANALYSIS (20 Points):
Consider this variation of the probabilistic counter problem: we have a counter C and for each operation `inc`(C), we will increment the counter value from i to $i+1$ with probability 2^{-i}, and leave the counter value unchanged at i with probability $1 - 2^{-i}$. Also, `look`(C) will return the value 2^i if the current value of C is i. (a) Describe the sample space Ω after we perform a sequence of $m \geq 1$ `inc`(C)'s. Let X_m be the random variable giving us the value of `look`(C) at the end of m `inc`(C)'s. (b) Describe X_m as a function with domain Ω. (c) Give a guess as to the expected value of X_m. Do not solve this exactly, but give some heuristic justification.

`ANSWER:` (a) $\Omega =$ ____

`A:`____

`ANSWER:` (b) X_m is ____

`A:`____

`ANSWER:` (c) $E[X_m]$ is ____

`A:`____

SOLUTION:
(a) $\Omega = \{0, 1\}^m$ (exactly as in hw5).
(b) $X_m : \Omega \to \mathbb{R}$ where $X_m(w) = 2^{\#(w)}$ and $\#(w)$ is just the number of 1's in the string w.

(v) MATRIX CHAIN (15 Points)
Let $(n_0, n_1, \ldots, n_5) = (2, 1, 4, 1, 2, 3)$. We want to multiply a sequence of matrices, $A_1 \times A_2 \times \cdots \times A_5$ where A_i is $n_{i-1} \times n_i$ for each i. Please fill in matrix in Figure 2.

SOLUTION:

	0	1	2	3	4	5
0		0				
1			0			
2				0		
3					0	
4						0
5						

Optimum Cost Matrix C

Figure 2: $C[i, j]$ is optimal cost to multiply $A_i \times \cdots \times A_j$.

	0	1	2	3	4	5
0	2	0	8	6	10	18
1		1	0	4	6	12
2			4	0	8	18
3				1	0	6
4					2	0
5						3

Optimum Cost Matrix C

Figure 3: SOLUTION

2. Depth-First Search (20 Points)
In the DFS Algorithm for searching a directed graph G, the nodes in G are initialized to "unseen" and when they are first encountered, they become "seen". Assume every node is either "seen" or "unseen" in our algorithm.
(i) (4 Points) Please give this DFS algorithm in its recursive form. You must explicitly show the node status change, from "unseen" to "seen", and explicitly add DFS Tree edges in your algorithm.
(ii) (16 Points) A path in G is called an "unseen path" if every node in the path is "unseen". Prove using induction the following: If there is an unseen path from node u to node v when u is first seen, then node v is a descendent of u in the DFS tree. HINT: Clearly state your inductive hypothesis. Do not assume indirectly what you have to prove (e.g., if you assume that the DFS algorithm will visit all reachable nodes and form a tree).
SOLUTION:
(i) The algorithm is as follows:

```
DFS($u$)
    for each node $v$ adjacent to $u$,
        if $v$ is "unseen", color it "seen"
        Add $(u, v)$ as an edge in DFS tree
        DFS($v$)
```

(ii) The inductive hypothesis is tricky to get correct! For instance, you cannot use induction based on the length of the shortest unseen path from u to v. This is because the DFS algorithm has its own order for visiting nodes, and your induction MUST refer to this order.

First, we need to define a total order on all paths from u to v: If a, b are two nodes adjacent to a node u and we visit a before b, then we say $a <_{dfs} b$ relative to u. If $p = (u, u_1, u_2, \ldots, u_k, v)$ and $q = (u, v_1, v_2, \ldots, v_\ell, v)$ (where $k, \ell \geq 0$) are two distinct paths from u to v, we say $p <_{dfs} q$ if $u_1 = v_1, \ldots, u_m = v_m$ and $u_{m+1} < v_{m+1}$ relative to u_m. Note that m is well-defined (in particular, $m < \min\{k, \ell\}$). Now, we define the DFS-distance between u and v to be the length of the $<_{dfs}$-least unseen path from u to v. Let $(u, u_1, \ldots, u_k, v)$ be the $<_{dfs}$-least unseen path from u to v.

INDUCTIVE HYPOTHESIS. Let IND(k) be the statement: if the $<_{dfs}$-least unseen path from u to v has length $k + 1$, then v is descendent of u in the DFS tree. Hence our goal is to prove IND(k).

First note an easy fact: if there NO unseen path from u to v at the time we visit u, then v is not a descendent of u in the DFS tree. Observe that is just the converse of what we need to prove. The contrapositive says: if v is a descendent of u then there is an unseen path from u to v at the time we first see u.

BASE CASE: If $k = 0$, then we need to prove IND(0). But $k = 0$ says that v is adjacent to u. Right after we color u as "seen", we will recursively call DFS(u). In the for-loop for DFS(u), we will add the edge (u, v) to the DFS Tree, and so v is a descendent of u.

INDUCTIVE CASE: If $k > 0$, then we will recursively call DFS(u_1). We claim that at this moment, the $<_{dfs}$-least unseen path from u_1 to v is $(u_1, \ldots, u_k, v)$. If this is not the case, that means that some $u_2, \ldots, u_k$ has become seen while we were computing DFS(u') where $u' <_{dfs} u_1$ and u' is unseen when we first visit u. By the "easy fact" (converse), we conclude that there is an unseen path from u' to v when we first see u'. Thus there is an unseen path from u to v when we first see u. This unseen path would be $<_{dfs}$ than $(u, u_1, \ldots, u_k, v)$, contradiction.

Thus our claim is valid. By IND($k - 1$), we conclude that v will be a descendent of u' in the DFS tree. Hence v is a descendent of u in the DFS tree.

3. Hashing (25 Points)
Consider the two ways to resolve collision by using chaining: coalesced chaining was described in Homework 5.
(i) (4 Points) Give an example where chains in separate chaining and coalesced chaining are different. For this question only, assume that in separate chaining, we insert new elements at the *end* of a chain. NOTE: chains must be viewed as a sequence of keys. You must ignore non-essential differences in the ways that a chain might be represented in separate chaining or in coalesced chaining.
(ii) (4 Points) Give an example to show that coalesced hashing breaks down if we do not have the DELETED flag. NOTE: DELETED flag is to be distinguished from the EMPTY flag.
(iii) (8 Points) Is it true that we waste space in coalesced chaining? Is it true that we waste time in coalesced chaining? Justify your answers.
(iv) (9 Points) Compare coalesced chaining and linear probing: give similarities and differences, pros and cons. Please mention the phenomenon of clustering. NOTE: assume

that in linear probing, the ith probe for a key x is give by $h(x, i) = h(x) + i$ (and not $h(x) + c.i$ for some $c > 1$). This makes the two methods comparable.

SOLUTION:
(i) Assume $h(x) = x \mod 10$ where x is an integer. Let $T[i]$ be the ith slot of our hash table. Suppose we insert the following sequence of keys: $1, 11, 2$. Then in separate chaining, we get two chains $C_1 = (1, 11)$ and $C_2 = (2)$. In coalesced chaining, we get one chain $C'_1 = (1, 11, 2)$. This is represented as: $T[1].Key = 1$ $T[1].Next = 2$, $T[2].Key = 11$, $T[2].Next = 2$, $T[3].Key = 2$, and $T[3].Next = -1$. Note that we can enter the chain C'_1 in the middle: e.g., from the viewpoint of key 2, its chain looks like $(11, 2)$
(ii) If we now delete key 11 from the table in part (i), and we do not mark it to be deleted, then when we lookup key 2, it will be wrongly reported as "not found"
(iii) Space is not wasted because DELETED slots will be used. That is, if the hash table has n slots, then we will be able to store n keys. On the other hand, time is wasted by the DELETED keys when when they appear in a chain, Compared to separate chaining, we will always spend at least as much time to lookup a key.
(iv) They are very similar in terms of how they find new empty or deleted slots for insertion. Note that both methods require the "DELETED" flag.

But the sequence of slots followed by Lookup in Coalesced Hashing can apparently be less than that for Linear Hashing. That is because we may be able to skip a block of occupied slots by following pointers. This coalesced hashing may be faster in its algorithms.

Coalesced hashing requires an extra field per slot, to store the pointers. So it is more wasteful in space.

Primary clustering appears in both methods, but its effects are less pronounced in coalesced hashing because of the ability to follow pointers.

4. NP-Completeness (20 Points)
(i) (2 Points) Recall the reducibility relationship between two languages, denoted $L \leq L'$, and read: L is reducible to L'. State the definition of reducibility.
(ii) (4 Points) Say a language $L \subseteq \Sigma^*$ is **trivial** if $L = \emptyset$ (empty set) or $L = \Sigma^*$. Which languages can you reduce to a trivial language?
(iii) (2 Points) Define the concept of an NP-complete language using the $\leq$-relation of (i).
(iv) (0 Points) Let $L \in NP$. You can assme that there is a deterministic Turing machine that accepts L.
(v) (12 Points) In your definition of reducibility in (i), there ought to be a requirement about "polynomial time". Suppose we omit this requirement (but keeping the rest). Write $\leq'$ instead of $\leq$ for this new notion of reducibility. In the definition of NP-completeness in (iii), suppose we use $\leq'$ instead of $\leq$. Such languages are said to be "funny NP-complete". Show that every non-trivial language in NP is funny NP-complete.

SOLUTION:

(i) Assume $L \subseteq \Sigma^*$ and $L' \subseteq \Gamma^*$. Then $L \leq L'$ means that there is a function $t : \Sigma^* \to \Gamma^*$ that can be computed by a deterministic Turing machine in polynomial time such that for all $x \in \Sigma^*$, $t(x) \in L'$ iff $x \in L$.
(ii) You can only reduce another trivial language to a trivial language.
(iii) L is NP-complete if (a) $L \in NP$ and (b) for all $L' \in NP$, $L' \leq L$.
(iv) Let $L \in NP$. Then there is a nondeterministic Turing machine N that accept L in time $T(n) = O(n^k)$ (for some constant k). We construct a deterministic Turing machine M that simulates N as follows: on input x, M
(v) The funny NP-complete languages is equal to all the non-trivial languages in NP. Let L be any non-trivial language in NP. If $L' \in NP$, we must show that $L' \leq' L$. Choose any word $w_0 \in L$ and any $w_1 \notin L$ (note that w_0, w_1 exists because L is non-trivial). Let M' be any nondeterministic machine that accepts L'. Consider the following deterministic Turing machine M: given any word $x \in \Gamma^*$ (Γ is the alphabet of L') M will simulate M' on input x. Although M' is nondeterministic, M is able to simulate it